AirShow CFD Software User's Guide

ROUGH DRAFT
Revised 10/5/04

Stan Mohler, Jr.
October 2004
QSS Group, Inc.
NASA Glenn Research Center
Cleveland, OH

Table of Contents

This work was supported by the NASA Glenn Research Center under contract NAS3-00145 with QSS Group, Inc..

1.0 Introduction

AirShow is a software application for the display of aerodynamic data associated with Computational Fluid Dynamics (CFD). Structured 3-D computational grids and computed aerodynamic flow data are visualized. AirShow can display block outlines to quickly reveal blocking arrangements, block numbering, and dimensions. Grid planes can be shown, colored uniformly or else according to aerodynamic quantities, including Mach number, static and stagnation pressures, temperatures, and densities. Components of velocity as well as the 5 "Q" variables of a PLOT3D solution file can also be shown.

The program reads binary PLOT3D grid and solution files. The user can interactively create and orient the displayed objects with the mouse. PLOT3D[1] is a program and a data format created by NASA for the processing and display of CFD data. For more information, read Appendix A of this guide or go to http://www.nas.nasa.gov/Research/Software/swdescription.html and scroll down to PLOT3D.

AirShow was created to provide a free tool that reveals the basic features of large PLOT3D grid and solution files as quickly as possible with the minimum necessary user input. AirShow opens large files much faster than FAST[2] (due to avoiding FAST's comprehensive IF-THEN tests that determine the minimum and maximum values of the grid and solution data read in). Before FAST will show the block outlines, the user must manually instantiate a grid plane in each block (or else write a script to do so). At that point, FAST will show the blocks with no easy way to identify the block number. AirShow labels each block in the graphics display. Another advantage of AirShow is avoidance of (1) FAST's laborious Calculator panel to compute each scalar function the user is interested in visualizing and (2) the time-consuming wait while the Calculator computes at every grid point in the data. AirShow, by contrast, replaces the Calculator panel with a simple drop-down menu for choosing the aerodynamic function to display on a selected grid plane. Upon the user's choosing of an aerodynamic function, AirShow computes the function only on the requested grid plane, similar to PLOT3D, thereby minimizing the amount of computation. AirShow is advantaged over PLOT3D by the latter's lack of a GUI.

Both FAST and PLOT3D have advantages over AirShow as well. Both applications are far richer in features than AirShow. Both can display many more aerodynamic functions than AirShow. Therefore AirShow is often best used as a good tool to augment other tools.

The Java and C++ source code for AirShow demonstrates one approach to creating sophisticated, fast, graphically intensive applications that are portable. The source code can serve as a reference application to bring the software developer up to speed on how to build a robust, graphically and numerically intensive application using Java technologies such as Swing, synchronized threads in Java, and the Java Native Interface (JNI) [3], as well as on interactive OpenGL 3D graphics.

AirShow makes use of Mark Kilgard's GLUT library[4] which is not in the public domain but whose source code is freely distributable without licensing fees. Both AirShow and GLUT are provided without guarantee or warrantee expressed or implied.

2.0 Description of Functionality

AirShow enables the user to easily toggle the visibility of the outlines of individual grid blocks (also called "zones"). The block numbers appear floating inside each outline, enabling the user to quickly understand the blocking structure of problems that have many grid blocks.

Like other CFD visualization packages, AirShow presents a graphical view in a window and GUI controls nearby. Figure 1 shows a screen shot of AirShow, including the Block Outlines panel used to control the display of block outlines.

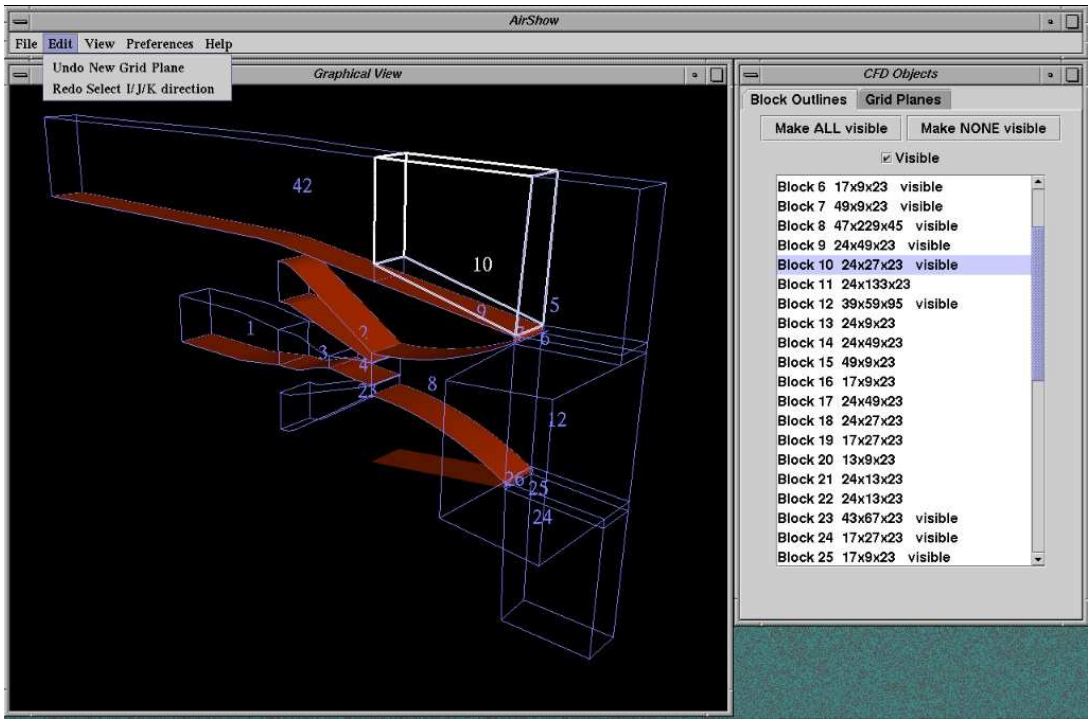

Figure 1. Display and control of block outlines

The user can easily bring into view grid planes from any block. These grid planes can be displayed as wireframes or surfaces, colored by a constant color or by one of several flow quantities (Mach number, static pressure, etc.).

Figure 2 shows the Grid Planes panel used to control the display of grid planes. Figure 3 shows a screen shot with grid planes for a commercial jet engine inlet.

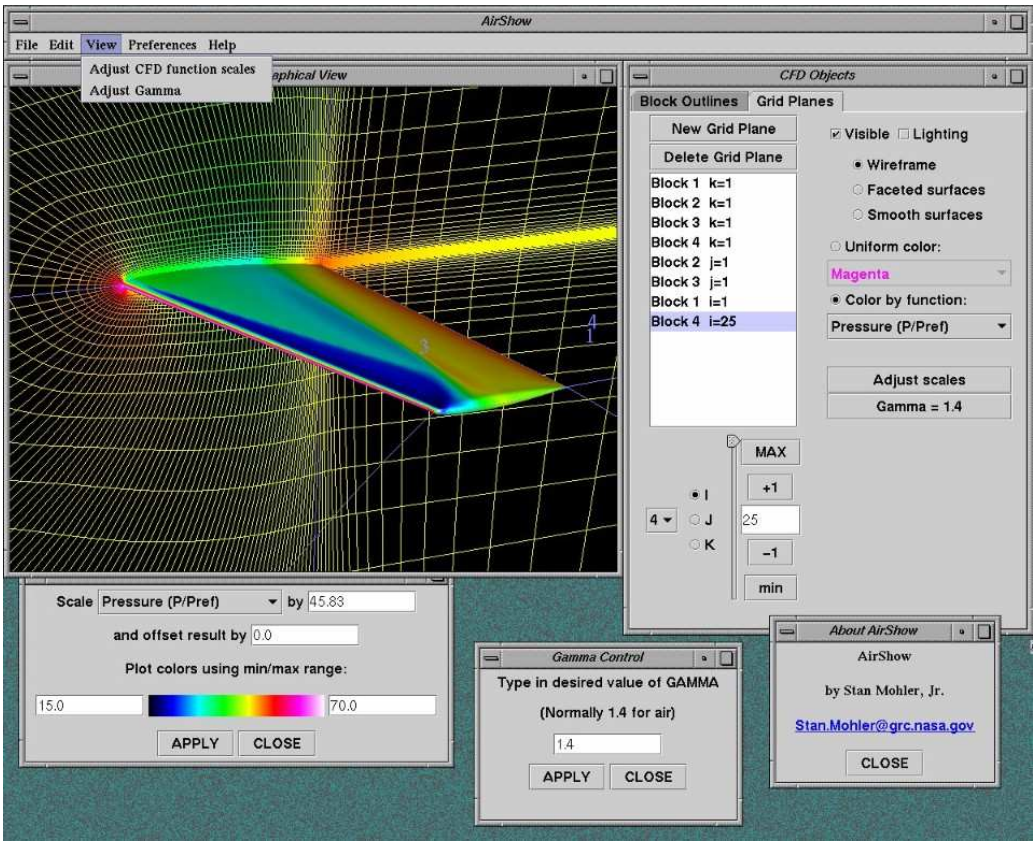

Figure 2. Display and control of grid planes

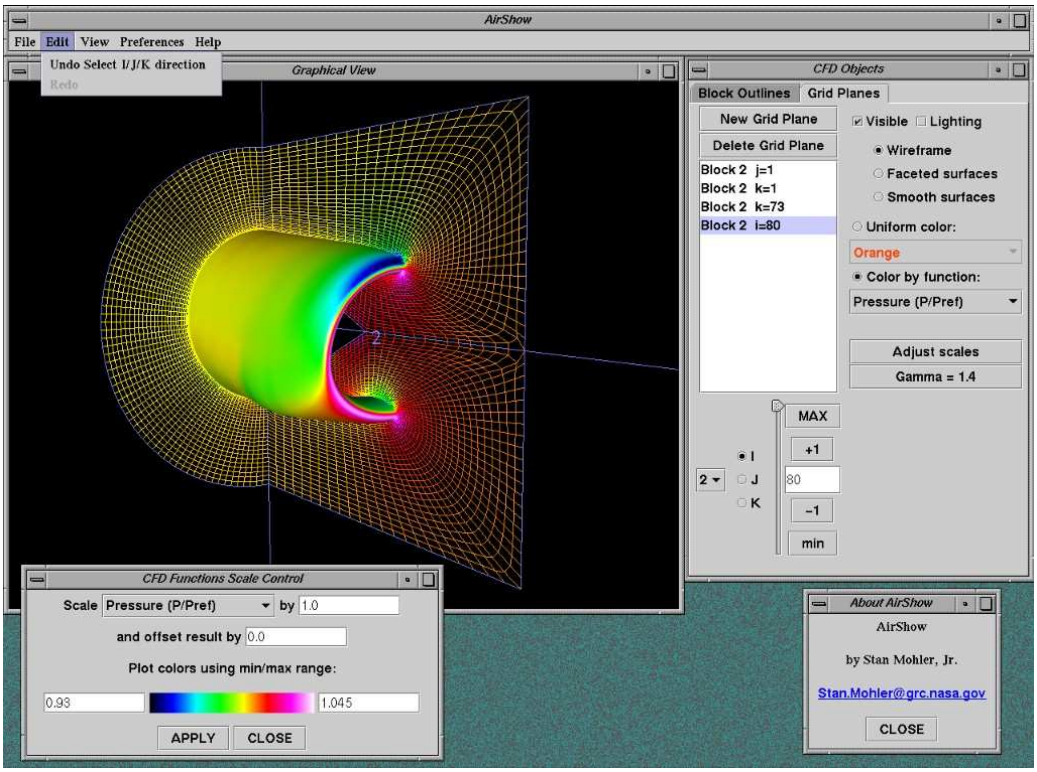

Figure 3. Display and control of grid planes (another example)

Many operations can be undone and redone, thanks to the use of the Java UndoManager and UndoableEditSupport classes[5].

A popup menu in the graphical view, operated by the right mouse button, selects the function of the left mouse button. Figure 4 shows the popup menu. Functions include rotation, translation, and zooming performed by holding down the left mouse button and dragging the mouse. The view perspective can also be reset. Another function is to set a new center of rotation by clicking on a point on one of the graphical objects. The coordinates of the point clicked on are printed to standard output. Finally, there is also the option of displaying the graphical objects as red and blue anaglyph wireframes suitable for 3D glasses.

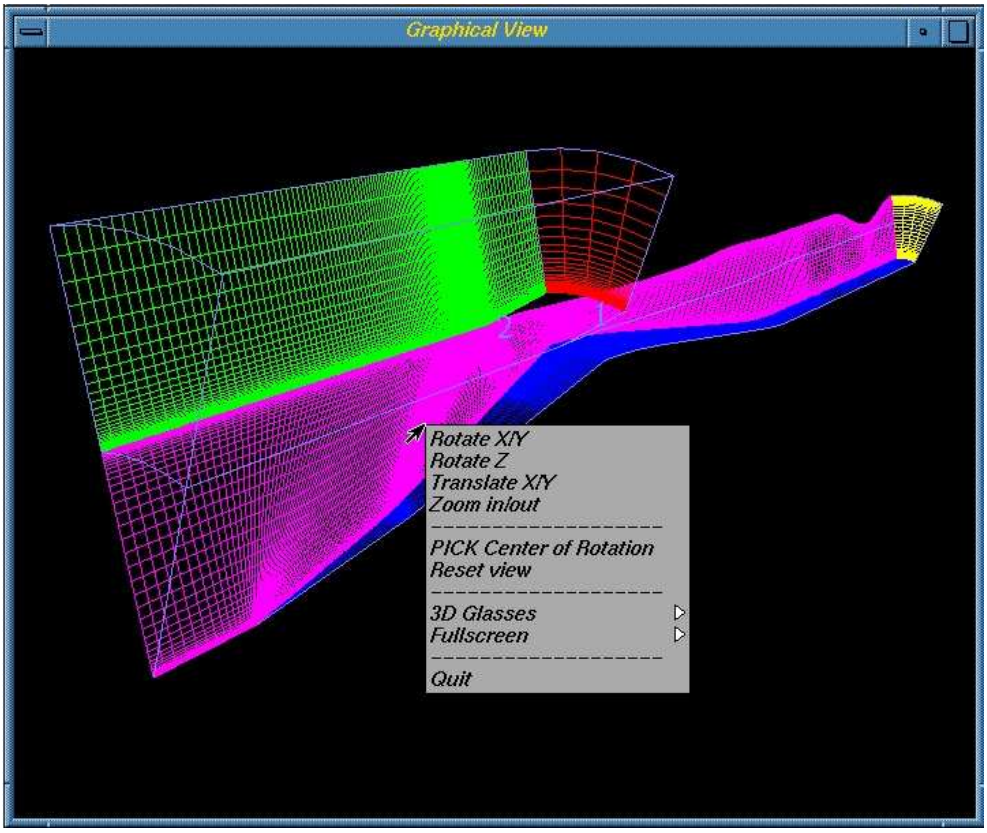

Figure 4. Popup menu in graphics display.

In developing the software, care was taken to produce aesthetically pleasing rotations, translations, and zooming. Rotations are about the screen axes rather than body-fixed axes. In addition, zooming and translation slow down as the eye approaches the center of rotation. Thus, by setting the center of rotation on some small feature in the model with a mouse click, the user can effortlessly zoom in on that feature without fear of suddenly overshooting it. Upon zooming back out to where the entire model is visible, dragging the mouse allows a comfortable rate of translation from side to side or up and down. Upon reorientation, including zooming far in to, or far from, the center of rotation, the clipping planes are automatically adjusted to keep the model in view.

3.0 Program Architecture

The overall architecture of AirShow follows the Model-View-Controller pattern. The Model consists of object classes representing the CFD data, i.e., grids and solutions contained in PLOT3D input files. The View provides a graphical window in which 3-D graphics are displayed to represent the model to the user. The Controller consists of the GUI that the user manipulates in order to adjust the view of the model. Each of the three components runs in its own synchronized thread and exchanges messages with the other components. Other patterns include the Command and Peer. The Command pattern provides undo/redo capability. The Peer pattern provides C++ classes that correspond to, and exchange messages with, some Java classes.

AirShow was implemented in the Java and C++ languages. Java was chosen for the main thread of AirShow for several reasons. Java threads provide a portable multithreading capability with synchronization. Java's Swing library provides sophisticated, cross-platform GUI components. Finally, Java can access C and C++ functions via the Java Native Interface (JNI).

AirShow depends on the ability to do three things as quickly as possible: read binary data files, perform intensive numerical calculations, and render complex 3-D objects. Currently, C and C++ are widely considered faster than Java for these three tasks. Therefore, the Model and View were largely coded in C and C++ in order to take advantage of the higher execution speed of the native machine code produced by those languages.

Figure 5 is a conceptual diagram showing how Java and C++ work together in AirShow.

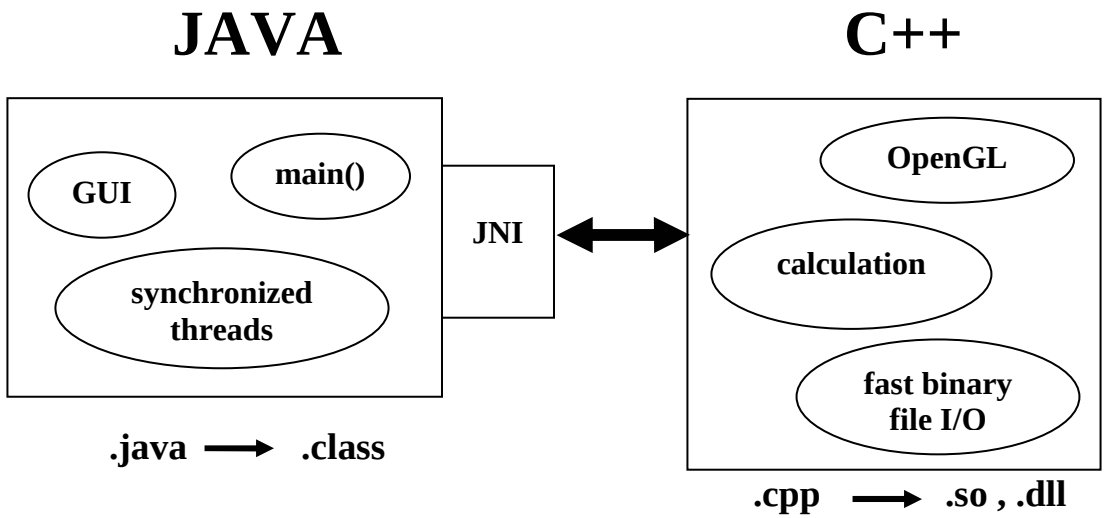

Figure 5. AirShow combines the best strengths of Java and C++ in one application

The Swing GUI is the program's main thread. This thread creates another Java thread for the View, which contains a call to the C-language GLUT function glutMainLoop() to process interactive graphics events such as object rotation and translation. These two threads exist for the life of the program. The main thread also creates temporary UpdateViewThreads to keep the View synchronized with the Model. A lock, implemented using Java's synchronized methods, ensures that an UpdateViewThread does not start modifying data until any previously created UpdateViewThread terminates.

While an UpdateViewThread runs, the Swing thread continues to process the user's input. The user never has to wait for a calculation to finish before choosing an alternative calculation. The GUI remains responsive at all times. If the user chooses to change the display, thus invalidating the current update, the UpdateViewThread will quickly notice and terminate itself to make room for the new UpdateViewThread. If the user initiates a calculation and then wishes that another calculation had been selected, he or she can immediately proceed to make new selections. The GUI will register the change, the thread performing the calculations will terminate itself, and the new calculation will start.

An example of these thread interactions is shown as a UML sequence diagram in Figure 6 where a user clicks on the "New Grid Plane" button. Pressing the button causes creation of a Java UpdateViewThread. This thread updates the data to be drawn in the graphics window. Just before terminating, this thread sends a message to the GLUT thread to redisplay the graphics scene. The GlutThread responds by calling OnRedraw(). Note that the GlutThread reacts to its own events, such as its timer, by calling callback functions, such as OnTimer(). GlutThread uses OnTimer() to periodically check for messages from other threads. The GlutThread is synchronized with the UpdateViewThread.

Figure 7 shows another sequence diagram, finer grained, where the user, having changed his mind, follows up the mouse click depicted in Figure 6 by immediately selecting a different grid block than was being prepared by AirShow. The invalidated UpdateViewThread is terminated and a new UpdateViewThread is allowed to start running.

Figure 8 is a sequence diagram showing how mouse events in the graphics window are processed. A user is rotating the graphical objects by clicking in the graphics window and then dragging the mouse. The GLUT thread continually calls OnRedraw() to increment the rotation angle and redraw the scene. In this way, rapid interactive graphics take priority in AirShow. If the user starts rotations while an UpdateViewThread is in the middle of changing the appearance of an object, the UpdateViewThread will quickly yield and the object's appearance (e.g., its coloring) will freeze for the duration of the user's mouse rotations of the scene.

There are important programming "gotchas" to be avoided when developing multithreaded programs with OpenGL and when using the Java Native Interface. Appendix C details some lessons learned while developing AirShow.

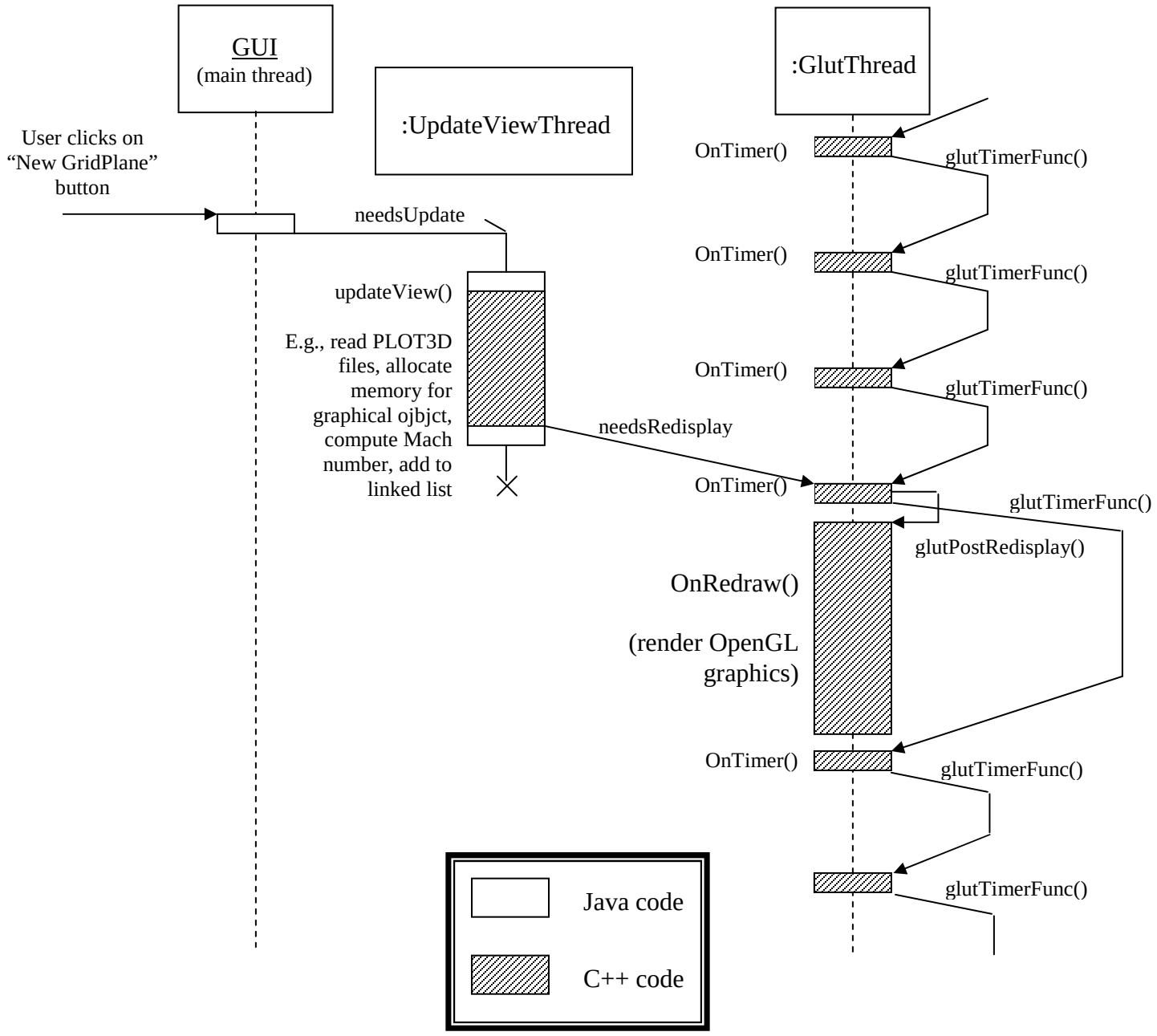

Figure 6. UML Sequence diagram showing a coarse-grained view of the three kinds of threads in AirShow

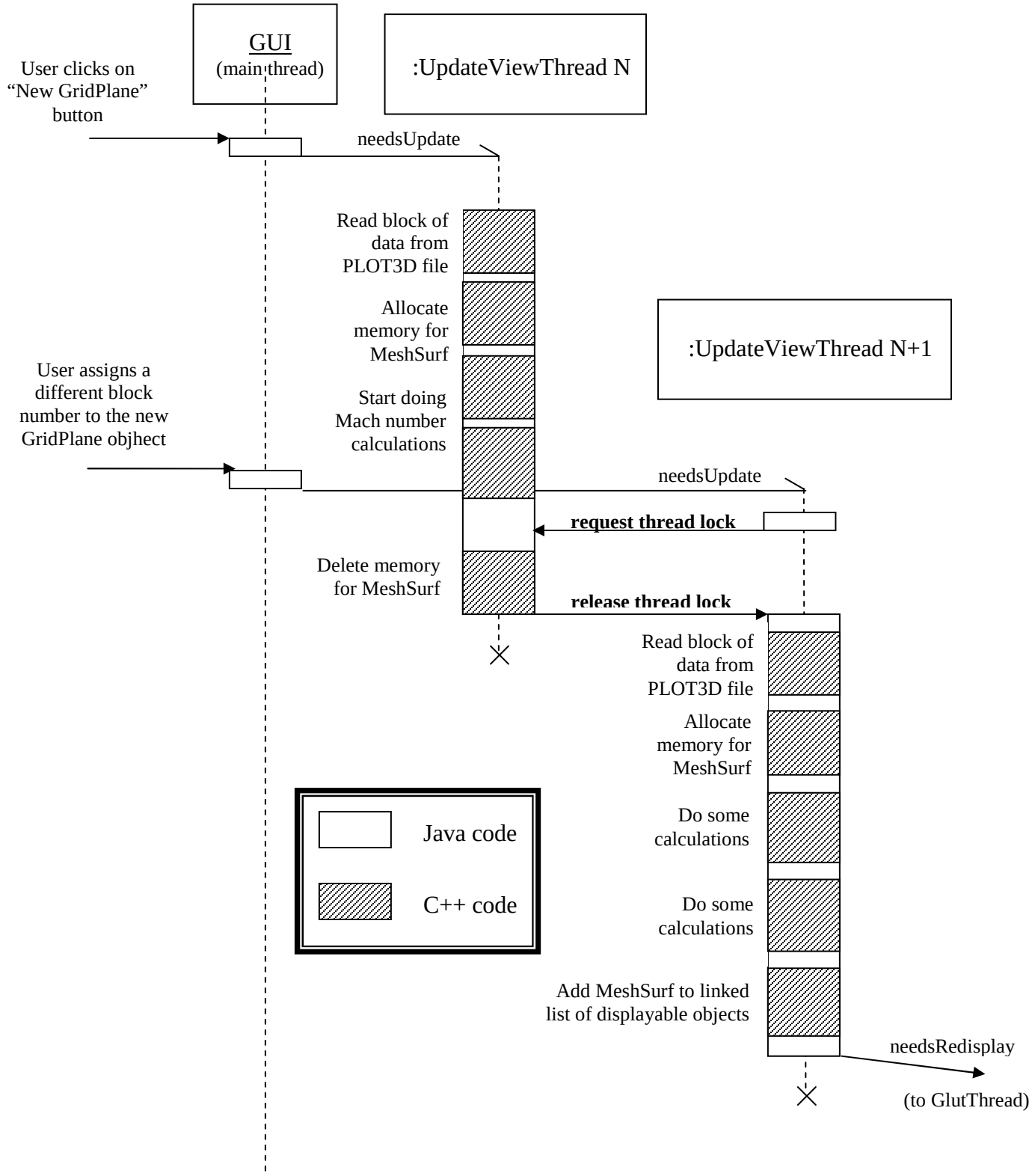

Figure 7. UML Sequence diagram showing fine-grained behavior of threads in AirShow. Here, a user has invalidated some selection before the invalid update is complete.

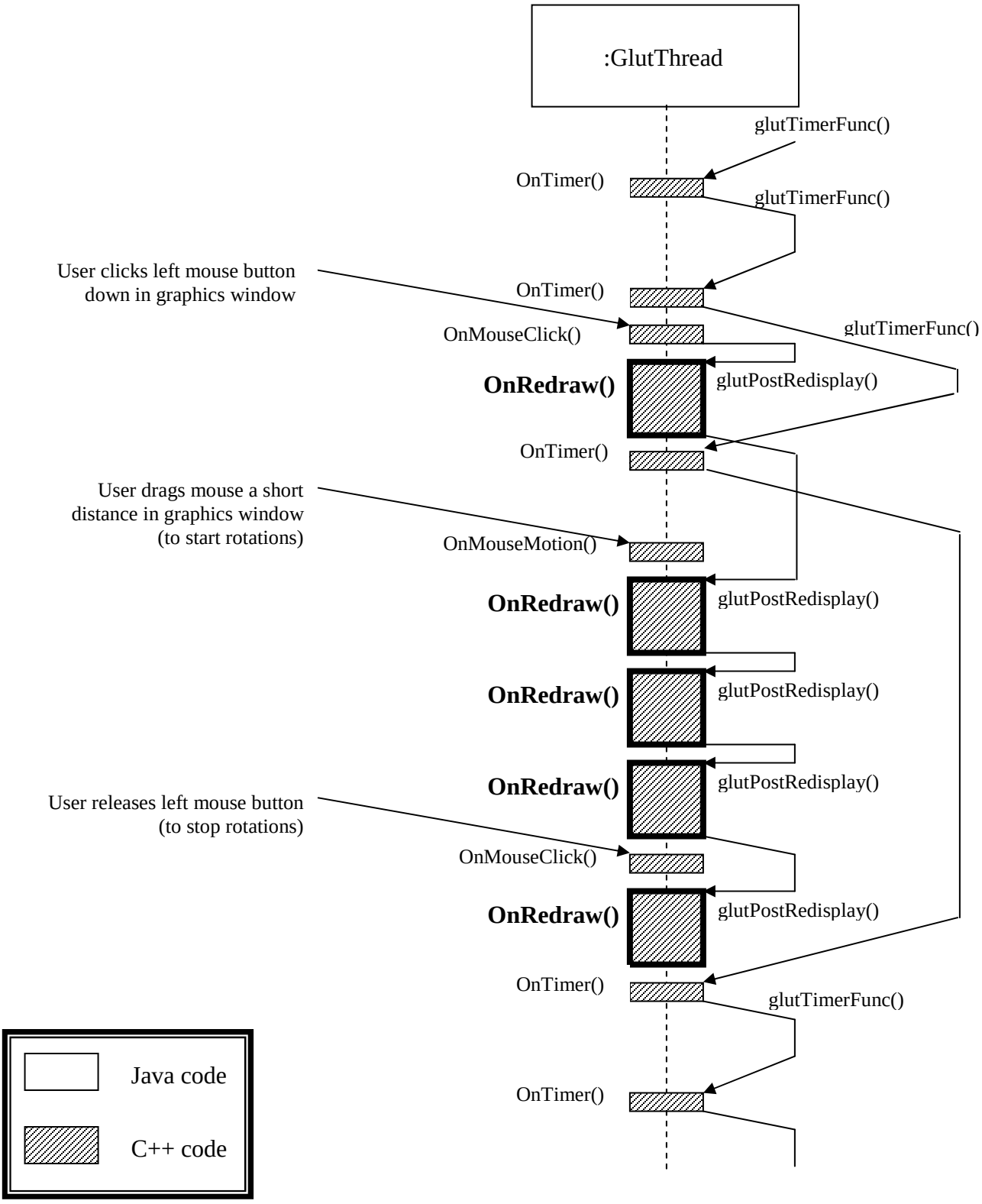

Figure 8. UML Sequence diagram showing how mouse events are processed in the graphics window.

The use of two languages was eased by the use of the Peer pattern. For example, a GridPlane class was separately written in both Java and C++. Each of the two classes contained attributes representing the block number, whether the grid plane was an I, J, or K grid plane, and the actual value of I, J, or K. The Java GridPlane object keeps the C++ peer updated via method calls. However, only the C++ peer class accesses the voluminous x, y, and z grid point coordinate data and the corresponding CFD flow data. These data are read from ordinary binary files using fast native system calls available to the C++ code on Unix, Linux, and Windows systems.

AirShow caches the grid coordinates of only one grid block at a time, leaving the rest of the data on disk in the binary PLOT3D file. This strategy enables AirShow to display multiple blocks of CFD datasets that would be too large to fit into physical memory simultaneously. When the user repeatedly increments the index of some grid plane, AirShow has immediate access to the necessary grid points in memory. When the user suddenly demands operations on some other grid block, necessitating AirShow to access data from a grid block that is not currently in cache, the cached grid block is deleted and a fast C read() function is called to quickly read in the new data from disk.

4.0 Installing the Software

4.1 Introduction

AirShow has been compiled and run on MIPS-based SGI computer systems running Irix, x86-based PC's running Linux, and Windows. It is probable that AirShow would compile and run on additional systems with little or no modification due to the portability provided by Java, standard C++, OpenGL, and the GLUT library.

The following software must be installed before AirShow can be run:

- AirShow (includes Java class files and native dynamic shared library)
- The OpenGL Utility Toolkit (GLUT) version 3.6 (or 3.7 on Windows) – provided with AirShow
- Java 2 Standard Edition Runtime Environment (JRE), preferably 1.4 or above
- OpenGL or Mesa

To compile AirShow yourself, you will need the following in addition to the above:

- Java 2 Standard Edition SDK (contains JRE), preferably 1.4 or above
- AirShow Java source files
- AirShow C++ source files (libAirShow.cpp and libAirShow.h)

As distributed, AirShow provides all of the above except for the Java2 runtime, the Java2 SDK, and OpenGL or Mesa. So the user must see that Java and OpenGL or Mesa are present.

4.2 Obtaining the Software

AirShow can be downloaded for free from the NASA Software Repository at https://technology.grc.nasa.gov/software/ . The user must create an online account and sign a Software Usage Agreement. Additional online information is available at the AirShow Web page at http://www.grc.nasa.gov/WWW/AirShow/ .

For Unix/Linux, AirShow is distributed as something like AirShow1.0.0.tar.gz where the "1.0.0" would be the version number. Similarly for Windows, AirShow is distributed as something like AirShow1_0_0.zip. Each file expands into a new directory called AirShow containing files and subdirectories.

On Unix/Linux, extract AirShow from the .gz file by the following two commands:

```
gunzip  AirShow1.0.0.tar.gz
tar xvf AirShow1.0.0.tar
```

On Windows, unzip the .zip file in the C:\ directory. If you don't have an unzip program, but you do have Java installed on your PC, then just run the following command in a shell:

```
jar xvf AirShow1_0_0.zip
```

These procedures will produce a directory called "AirShow" inside your current directory. If you already have an AirShow directory, it will be overwritten. So you might want to rename the old one as "AirShowOld" or something.

The AirShow/ directory contains a README file. It also contains an empty file named something like "VERSION_x.y.z" to indicate the version of AirShow you have there.

The following subdirectories exist:

Misc/	...screen shots and a user guide
TestCases/	...sample PLOT3D files to read into AirShow
glut/	...Mark Kilgard's GLUT 3D graphics library
javaClasses/	...Java class files for AirShow
javaDoc/	...javadoc-generated documentation for Java classes
javaSrc/	...Java source code for AirShow
libAirShow/	...AirShow dynamic libraries with C++ source code

resrc/	...resource files used by AirShow
utilities/	...some small programs and scripts to do such things as convert PLOT3D files between unformatted, formatted, and binary.

If your system already has Java and OpenGL, then you may now be ready to run AirShow. However, complete instructions are provided below for obtaining Java, OpenGL, and GLUT from the Web. Note however that AirShow is provided with the GLUT library in the AirShow/glut-3.6/ directory. So you should not need to download or install GLUT. You may need to compile it, however.

The Java environment (JRE and SDK) can be downloaded for SGI machines for free from http://www.sgi.com. For other platforms, go to http://java.sun.com/j2se/ for free download. Linux distributions should come with the Java JRE and SDK.

OpenGL is preinstalled on SGI machines, and commonly on Linux. On Windows, OpenGL should be available via files named Opengl32.dll and Glu32.dll inside C:\Windows\System\ or equivalent. Video cards that support 3-D hardware acceleration of OpenGL graphics may come with a special version of the DLL files or other OpenGL driver, on CD or disk, that is preferable. Linux users may find special OpenGL drivers for their video card included with their card or perhaps downloadable from the Web. Further information on OpenGL and graphics cards can be obtained from http://www.opengl.org. Users who will build AirShow themselves on Windows require two additional OpenGL files: Glu32.lib and Opengl32.lib .

An alternative to OpenGL is Mesa, available at http://mesa3d.sourceforge.net .

The OpenGL site provides links to the GLUT library, which can be downloaded for free. As of March 2004, GLUT was available at http://www.opengl.org/resources/libraries/glut/glut_downloads.html. SGI and Linux users need to obtain (or build) the file libglut.a while Windows users need Glut32.lib and Glut32.dll. Note that SGI supports 3 incompatible types of executable code: o32, n32, and n64 Application Binary Interfaces. The instructions in this document describe building an n32 version of AirShow. Therefore, the n32 version of libglut.a is used.

Once the items you require from above are downloaded, proceed to install Java, OpenGL and GLUT using the instructions that come with the various packages. The GLUT library can be installed on an SGI system by placing the n32 version of file libglut.a into the directory /usr/lib32/. On Linux, you should probably put libglut.a into /usr/lib/GL. However, as stated above, AirShow is distributed with the GLUT files in the AirShow/glut-3.6/ directory where they can stay.

On Windows, you could install GLUT by obtaining glut32.lib and placing it in a location such as C:\Program Files\Microsfoft Visual Studio\VC98\Lib. Then obtain glut.h, and place it somewhere like C:\Program Files\Microsfoft Visual Studio\VC98\Include\Gl. However, as distributed, AirShow provides these files inside the AirShow\glut\win32\

directory. The Glut32.dll file also appears in C:\AirShow so that, when run in that directory, AirShow will find Glut32.dll.

Note that MS Developer Studio/Visual C++ comes with OpenGL header and library files.

4.3 Compiling AirShow

As distributed, AirShow is already compiled for SGI/MIPS, Linux/x86, and Windows. Therefore you may not need to compile. All the compile scripts are provided, and can be run as follows:

1. Go into javaSrc/ (or javaSrc\ on Windows).

2. Compile the Java source files by examining, possibly modifying, and then running the one appropriate script for your platform from the following three. The script you run may need an environment variable, such as $JAVA_HOME, modified for your particular system:

    compile.linux
    compile.sgi
    compile.bat

3. Go into javaClasses/.

4. Possibly modify and then run the one appropriate script from the following three to generate C-style header files used to enable communication between Java and C++:

    runJavah.linux
    runJavah.sgi
    runJavah.bat

5. If you are not on an SGI MIPS system, and not on PC Linux, and not on Windows, you must go into the glut/glut-3.6 directory and compile GLUT for your system. Read Mark Kilgard's README file there.

6. If you are on a Linux or SGI platform, go into libAirShow/. Modify and then run the one appropriate script to compile the libAirShow C++ code into a DSO named libAirShow.so:

    compile.linux
    compile.sgi

7. If you are on Windows, go into libAirShow/VC++/AirShow/. Open the MS Visual C++ project file (AirShow.dsw), and compile. The AirShow.dll file will be created in the Release directory.

The VC++ project file contains necessary settings which were set in the VC++ version 6 IDE like so:

1. Click on Build -> Set Active Configuration -> Win32 Release

2. Click on Project -> Add to Project -> Files: libAirShow.cpp, *.h

3. Click on Tools -> Options -> Directories -> Show Directories For: Include Files. Add in C:\J2SDK1.4.2_03\INCLUDE (or something appropriate for your system), C:\J2SDK1.4.2_03\INCLUDE\WIN32 (or something appropriate), and C:\AIRSHOW\GLUT\WIN32\INCLUDE .

4. Click on Tools -> Options -> Directories -> Show Directories For:->Library Files. Add in C:\AIRSHOW\GLUT\WIN32. This links in the files glut32.lib, glu32.lib, and opengl32.lib. Note that this linkage specification must be done separately for Debug and Release.

5.0 Running AirShow

5.1 Starting It Up

Run AirShow by executing the appropriate run script in the AirShow directory. Scripts are provided for SGI-MIPS, Linux/x86, and Windows, appropriately named run.sgi, run.linux, and run.bat. Make sure the script sets the JAVA_HOME environment variable properly for your system. These scripts set the Java CLASSPATH environment variable then start up the Java virtual machine to run AirShow. For further details, examine the scripts yourself. Note the particular options used with the java command.

Feel free to copy the run script to other directories and modify it as appropriate. On Unix/Linux, your $HOME/bin directory would be a good location, provided you have one in your path. Make sure the script sets the $AIRSHOW_HOME variable properly, most likely to $HOME/AirShow. Renaming the script “airshow” would be convenient. Upon logging back in to your computer, you would then be able to run AirShow from any directory by typing the command `airshow`.

Upon seeing the AirShow main menu bar, you should be able to open files inside the TestCases directory. However, your own binary PLOT3D files can be opened as well.

5.2 Preparing Input Files

Make sure there is a PLOT3D grid file available in the correct format. A solution file is optional. The PLOT3D files must be binary, 3D, multiblock, and non-IBLANKED. See Reference 1 for details.

Given formatted or FORTRAN unformatted PLOT3D files, PLOT3D itself can be used to output binary files using the following example PLOT3D commands:

```
list/binary/output=mygrid.bin
xyz
list/binary/output=mysolution.bin
q
```

PLOT3D can also convert between formatted and unformatted.

Users running the Wind-US CFD flow solver can use the CFPOST utility to convert their .CGD and .CFL files to PLOT3D binary files using something like the following which is for SGI machines:

```
! CFPOST SCRIPT
grid yourfile.cgd
solution yourfile.cfl
units inches
subset i all j all k all
zone 1 to last
plot3d x newfile.xyz.bin q newfile.q.bin iris
quit
```

Note that in general, unlike for ASCII text files, CGD files, and CFL files, a binary data file on one computer platform cannot simply be copied to another type of platform and be useable. AirShow comes with a command-line utility called fmt2bin that converts a FORTRAN formatted (i.e., ASCII) PLOT3D grid or solution file to binary. Fmt2bin.c compiles on SGI, Linux, and Windows.

5.3 Reading PLOT3D Input Files

Upon successfully starting AirShow, several status messages will appear on stdout, followed by the AirShow main menu.

The main menu should be easy to figure out. The user proceeds by clicking on File. Figures 7 through 9 show example screen shots.

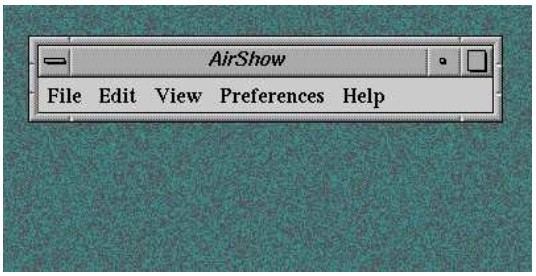

Figure 8. Initial AirShow appearance

Figure 9. Clicking the File menu

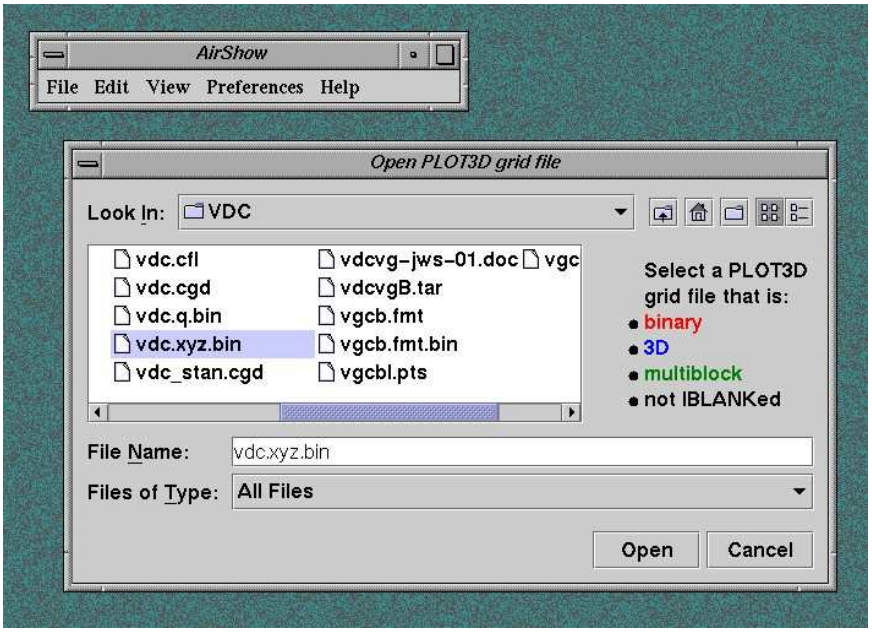

Figure 10. The file chooser for opening a PLOT3D grid file

5.4 Creation and Display of CFD Objects

There are two types of "CFD objects" currently implemented in AirShow: block outlines, and grid planes. A frame provides access to both types via two tabbed panes. Upon startup, all the block outlines are visible and one grid plane is visible corresponding to block 1 k=1. The grid plane is displayed as a lighted, shaded surface. Note that if the surface happens to be perpendicular to the user, it will appear white because the source of the light is modeled geometrically as if it were located between the user's eyes and shining into the computer screen. The lighting can be turned off with a mouse click.

The Block Outlines tabbed pane is visible by default. Controls are provided to turn visibility on and off for the individual block outline selected on the list, as well as a control to turn all block outlines on or off. When the user selects a block by clicking on its name in the list, that block will appear highlighted in the graphics display. A different block can be highlighted by simply selecting another block on the list. To turn off highlighting, simply make the current block outline invisible and then visible again, using the visibility check box.

The Grid Planes tabbed pane provides controls to turn lighting on and off, create and delete grid planes, switch between I, J, and K, increment/decrement the index value, switch the block number, etc. Many of the grid plane operations can be undone and redone via the Edit menu on AirShow's main frame.

Additional grid plane controls will appear if the user opens a PLOT3D solution file. These controls enable grid planes to be colored by aerodynamic quantities such as Mach number, static and stagnation pressure, temperature, and density, as well as the u,v, and w flow velocity components. The PLOT3D Q variables are also selectable. The scale for plotting, as well as the ability to redimensionalize the quantities, appear upon pressing the Adjust Scale button. Unlike the way some other visualization packages do things, the scale used for plotting any given flow quantity (e.g., Mach number) is applied consistently across all objects colored by that flow quantity.

To reorient the objects displayed in the graphics window, place the mouse pointer in the graphics window, hold the left mouse button down, and drag. To select another type of transformation, hold the right mouse button down and use the resulting pop-up menu. Choices include rotation about the screen x, y, and z axes, translation, and zooming. On Unix-like systems, dragging the mouse while holding down the middle of 3 buttons immediately zooms in or out. One can also pick a new center of rotation. To restore the initial orientation, one can choose to reset the view.

6.0 Conclusion

References

[1] Walatka, P.P., Buning, P.G., Pierce, L., Elson, P.A., "PLOT3D User's Manual", NASA TM 101067, March 1990

[2]Walatka, P.P., Clucas, J., McCabe, R.K., Plessel, T., Potter, R., "FAST User Guide", NASA RND-93-010, June 1993

[3] Sheng Liang, *The Java Native Interface Programmer's Guide and Specification*, Sun Microsystems, 2002

[4]Kilgard, Mark J., *The OpenGL Utility Toolkit (GLUT) Programming Interface – API Version 3*, November 1996

[5]Meshorer, T., "Add an Undo/Redo function to your Java Apps with Swing", *Java World*, June 1998

Appendix A. Description of PLOT3D File Format

See Reference [1] for details about the various PLOT3D file formats. The particular PLOT3D format used by AirShow is 3D, multiblock, "whole", non-IBLANKED, single precision binary for grids and flow solutions.

A grid file starts with an integer expressing the number of grid blocks. Next are three integers telling the I/J/K dimensions of the first grid block. Additional groups of three integers follow, one for each remaining grid block. At this point, the PLOT3D header data is complete, and the grid coordinates follow. There is a group of floating point numbers for each grid block. For each grid block, all the x-coordinates appear first in the order resulting from varying the I index the fastest and the K index the slowest. The x-coordinates are followed by the y- and then the z- coordinates.

A solution file has an identical header to the grid file, duplicating the dimensional data. A solution file is similar to a grid file, except x,y,z floating point numbers are replaced with the Q1, Q2, Q3, Q4, Q5 PLOT3D solution variables defined by equations in Reference 1. Appendix B can be used to clarify the meanings of the Q variables as well. Immediately before a grid block's Q data, each grid block has its own header consisting of the 4 floating point variables FSMACH, ALPHA, RE, and TIME. These four represent the free-stream Mach number, the angle of attack, the Reynolds Number, and the iteration number of the flow solver that produced the solution file.

Appendix B. Equations for CFD Flow Quantities used by AirShow

The following C++ code comes straight out of the libAirShow.cpp source file, and shows how the 5 PLOT3D file "Q" variables are used to compute the 14 aerodynamic quantities that AirShow displays:

```
float Calculator::getScalarFunctionValue(int code) {

 //  0 = Mach number
 //  1 = static pressure ( P/Pref )
 //  2 = stagnation pressure ( P0/Pref )
 //  3 = temperature  ( T/Tref )
 //  4 = stagnation temperature ( T0/Tref )
 //  5 = density  (q1)
 //  6 = stagnation density ( rho0/rhoRef )
 //  7 = q2
 //  8 = q3
 //  9 = q4
 // 10 = q5
 // 11 = u
 // 12 = v
 // 13 = w


 funcVal[5]  = q1;
 funcVal[7]  = q2;
 funcVal[8]  = q3;
 funcVal[9]  = q4;
 funcVal[10] = q5;

 rho = q1;
 float oneOverRho = 1.0/rho;

 u = q2 * oneOverRho;
 v = q3 * oneOverRho;
 w = q4 * oneOverRho;

 funcVal[11] = u;
 funcVal[12] = v;
 funcVal[13] = w;

 V2 = u*u + v*v + w*w;
 V = sqrt(V2);

 e0 = q5 * oneOverRho;
 e = e0 - 0.5*V2;

 p = gamma*(gamma-1.)*rho*e;

 funcVal[1] = p;

 T = p * oneOverRho;

 funcVal[3] = T;
```

```
 a = sqrt(T);

 Mach = V/a;

 funcVal[0] = Mach;

 fac = 1.0 + 0.2*Mach*Mach;

 T0 = T*fac;

 funcVal[4] = T0;

 p0 = p * fac*fac*fac*sqrt(fac); // Pstag = P*(1+0.2*Mach)^3.5

 funcVal[2] = p0;

 rho0 = rho * fac*fac*sqrt(fac); // rhoStag = rho*(1+0.2*Mach)^2.5

 funcVal[6] = rho0;

 return funcVal[ code ];
}
```

Appendix C. Some Programming Lessons Learned While Developing AirShow

While developing AirShow, several critical lessons were learned about the Java Native Interface, OpenGL graphics, and machine architecture:

- Designate only one thread to make OpenGL (and therefore GLUT) function calls. These functions are not thread-safe. If multiple threads make such calls, the Unix/Linux X server will produce "async reply" errors or worse. If other threads need to manipulate the graphics, then have those threads set variables visible to the OpenGL thread.

- On Unix/Linux, other crashes due to improper use of the X server were avoided by having Java load the native library from within the run() method of the thread designated to call the native code containing OpenGL calls. This thread was other than the main Java thread (which created Swing and AWT components).

- AirShow is extremely stable and should not crash. But during development, if a thread running native code had a segmentation fault due to bad pointers, the Java Virtual Machine would crash with voluminous and very cryptic messages. The inclusion of Unix signal handlers in the C++ code proved very helpful in identifying where the errors occurred.

- On Unix/Linux, make sure the libAirShow.so dynamic shared object was compiled for your exact architecture. Using a library compiled for the newer SGI MIPS4 instruction set caused AirShow to crash with a segmentation fault when run on an older MIPS2 SGI or else with the wrong choice of o32, n32 or n64 ABI.

- When passing a jstring argument to native code via the Java Native Interface, make sure to call the (*env)->ReleaseStringUTFChars() function once your native code is done with the jstring. This allows the Java VM to free the memory used by the jstring. Failing to do so causes a memory leak and possible instability in the application.

- The Java Native Interface does not support the C++ virtual keyword. Therefore you cannot use polymorphic inheritance in C++.

- The following two articles in the Sun Microsystems online JDC TechTips newsletter describe a potential problem when writing Java programs using the Swing GUI classes. They explain how to avoid a common but rarely noticed error when calling a GUI component's setVisible() method. The articles are dated Dec. 8, 2003 and Jun. 11, 2004:

  http://java.sun.com/developer/JDCTechTips/2003/tt1208.html#1
  http://java.sun.com/developer/JDCTechTips/2004/tt0611.html#1